Mesh Adaption and Automatic Differentiation for Optimal Shape Design(Domain Decomposition Methods and Related Topics)

AUTHOR(S):

Mohammadi, Bijan

CITATION:

Mohammadi, Bijan. Mesh Adaption and Automatic Differentiation for Optimal Shape Design(Domain Decomposition Methods and Related Topics). 数理解析研究所講究録 1997, 989: 78-91

ISSUE DATE:

1997-04

URL:

http://hdl.handle.net/2433/61070

数理解析研究所講究録
989巻 1997年 78-91

Mesh Adaption and Automatic Differentiation for Optimal Shape Design

Bijan Mohammadi
INRIA, BP 105, 78 153 Le Chesnay, France
Bijan.Mohammadi@inria.fr

Abstract

A new approach for optimal shape design based on a CAD-free framework for shape and unstructured mesh deformations, automatic differentiation for the gradient computation and mesh adaption by metric control is presented. The CAD-free framework is shown to be particularly convenient for optimization when the mesh connectivities and control space size are variable during optimization. Constrained optimization for transonic regime has been investigated.

Keywords: Shape Design, Unstructured Mesh Adaption, Automatic Differentiation.

1 Introduction

Local mesh adaption by metric control is a powerful tool for getting mesh independent results at a reduced cost [1, 2, 3]. However, mesh adaption has never been used in optimization procedure because it implies a change of the design space and mesh connectivities after each adaption and this is difficult to take into account. Moreover, these procedures are not differentiable at all. Therefore, optimization is usually performed on meshes with fixed connectivities and on a given control space (i.e. no control parameters are created during optimization). However, as for direct problems, the need for adapted meshes is clear in design processes.

The aim of this paper is to show how to couple mesh adaption and shape optimization tools. Here, not only the mesh in the computational domain, but also the number and repartition of the control parameters over the shape will change during optimization. Therefore, general shape and unstructured mesh deformation tools are necessary. In particular, these tools have to be as much as possible CAD-free, in the sense that: *the only geometrical entity available during optimization should be the mesh.* Another reason which justifies this requirement is that CAD-based shape deformation tools are quite complicated to use and also difficult to take into account when evaluating the Jacobian. In [4] we showed that we can consider all the mesh points on the body (in both 2 and 3D) as control points as far as we preserve the initial local regularity of the shape during the design process. This makes possible and easy to consider all kind of shapes.

As we use a gradient method in our shape optimization, it is necessary to have a good evaluation of the Jacobian of the 'discrete' cost function with respect of control

parameters. It is clear that if the number of control parameters is large, an adjoint method is necessary [5, 6]. But classical adjoint methods are often built in continuous level, after a linearization of the governing equations, and do not take into account for the inconsistencies of the numerical schemes (like numerical dissipation). In this work, we use automatic differentiation in reverse mode (AD) to produce the Jacobian of the discrete cost function. Hence, the non-consistencies of the numerical schemes are naturally taken into account in the Jacobian. Moreover, the cost of this evaluation is independent of the number of control parameters as in a classical adjoint method. Using this approach, we have studied [4, 7, 8] the impact of using different gradients based on different numerical fluxes (Roe and Osher with and without a MUSCL type second order reconstruction [14, 17, 15]) used for the solution of the Euler equations, on the optimization procedure. We also showed that this approach does not suffer from a change in the nature of the equations (from parabolic to hyperbolic). Viscous turbulent configurations have also been considered by tacking into account of a $k-\varepsilon$ two-equation turbulence model [9] and special wall-laws (including pressure and convection effects [11, 10]) in the Jacobian. We noticed that in an optimization procedure involving mesh deformation, wall-laws are more suitable than low-Reynolds modeling (i.e. solving the flow up to the wall) as this former approach requires much finer meshes for which conformal mesh deformation are hard to achieve. In particular, we showed that once the difficulties coming from the nonlinearities of the operators treated, it is easier to perform optimization for a turbulent flow configuration than for the corresponding inviscid flow (i.e. at the same Mach number).

As this work includes several tools, only a short description of each of them is given. The governing equations and the flow solver are described in section 2. Section 3 is devoted to the shape and mesh deformation tools. A short description of our mesh adaption approach is given in section 4. Our adaptive optimization algorithm and related details including the Jacobian computation by AD in reverse mode are described in section 5. Section 6 shows various results of adaptive shape optimization from transonic regime over airfoils.

2 Flow equations and solver

We denote by $J(x, U(x))$ the cost function we want to minimize under geometric and aerodynamic constraints. Here, x indicates the geometrical description of a configuration and the flow pattern ($U(x)$) around this shape is the solution of the steady Euler system of fluid dynamics in conservation form:

$$\nabla.(F(U)) = 0, \tag{1}$$

where U is the vector of conservative variables (i.e. $U = (\rho, \rho\vec{u}, \rho(C_v T + \frac{1}{2}|\vec{u}|^2))^t$) and F represents the advective operator. This system has 4 equations in 2D and the system is closed using the equation of state $p = p(\rho, T)$. For seek of simplicity, we only consider the Euler equations but the extension is already available for viscous turbulent flows [4, 7].

The flow solver is based on a finite-volume-Galerkin approach on unstructured triangular meshes (tetrahedra in 3D). A Roe [14] approximate Riemann solver has been used together with MUSCL reconstruction and Van Albada limiters [15]. The time dependent

equation

$$\frac{\partial U}{\partial t} + \nabla.F(U) = 0,$$

is marched in time to a steady state. Our time discretization is based on a 4-stage Runge-Kutta scheme. Inflow and outflow boundary conditions are of characteristic type [16] and for outflow boundaries care has been taken to correctly treat subsonic configurations. More technical details can be found in [12, 13].

3 Shape and mesh deformation tools

We describe the set of tools needed for shape and mesh deformation from a variation of control parameters $(x_c(n_c))$:

$$\delta x_c \rightarrow \delta x_w \rightarrow \delta x_m,$$

where $x_w(n_w)$ denotes the discretization points on the geometry and $x_m(N)$ the internal mesh nodes.

For 2D applications, control points are usually fitted by splines. Splines have two features:

1) They permit $n_c << n_w$.

2) They smooth the variations of control points when propagating to body discretization points.

But in this work, following the solution (through a local metric), the control space changes during optimization as points may be created or removed on the body. Therefore, the positions of the points have to be redefined after each adaption over the shape. Of course, this is possible in 2D but much more difficult in 3D. In all case, this increases the difficulty by introducing another layer in the adaptive optimization. Moreover, general 3D surface splines are quite complicated to handle, especially on unstructured meshes. To avoid 3D difficulties and reduce the dependencies between two adaptions, we introduce the following general framework for shape deformation in 2 and 3D:

1. All the nodes on the shape are control points (i.e. $n_c = n_w$).

2. To avoid oscillations, a smoothing operator is defined over the shape. This can be, for instance, a few 'local' Jacobi iterations to solve the following system:

$$(I - \varepsilon\Delta)\delta\tilde{x}_w = \delta x_w, \tag{2}$$

where $\delta\tilde{x}_w$ is the smoothed shape variation for the shape nodes and δx_w is the variation given by the optimization tool. By 'local' we mean that if the predicted shape is locally smooth, it remains unchanged during this step.

In the past, we used this framework for optimal shape design in 2 and 3D [8, 7, 4, 29] for cases with fixed connectivity and control space. We showed that configurations involving several thousands of control points can be easily treated.

The importance of the smoothing step above can be understood by the following argument:

We want the variation $\delta x_w \in C^1(\Gamma)$, if Γ designs a manifold of dimension $(n-1)$ in a domain $\Omega \in R^n$. From Sobolev inclusions, we know that $H^n(\Gamma) \subset C^1(\Gamma)$. It is easy to understand that the gradient method we use do not produce necessarily $C^1(\Gamma)$ variations

δx_w and therefore we need to project them into $H^n(\Gamma)$ (an example of this is given in picture (1)). This means that the projected variations $(\delta \tilde{x}_w)$ is solution of an elliptic

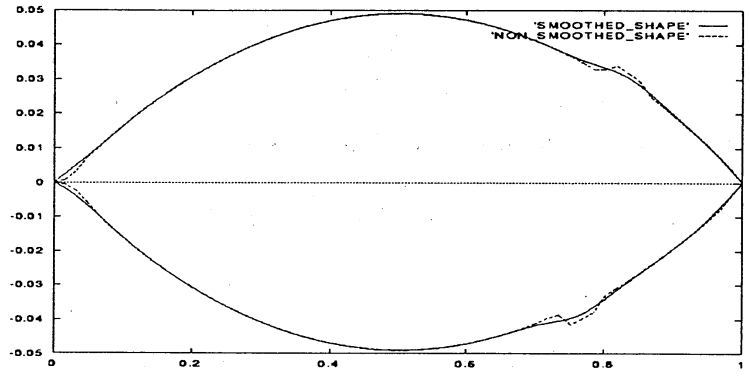

Figure 1: *Smoothed and unsmoothed shapes. We can see that the gradient jumps through shocks and also produces a non-smooth shape in leading edge regions. This is the result of the first iteration of the optimization. If we keep continue the shape becomes more and more irregular.*

system of degree n. However, as we are using a P^1 discretization, a second order elliptic system is sufficient even in 3D because the edges of the geometry (like a trailing edge) are considered as being a geometrical constraint for the design. Therefore we project the variations (δx_w) only into $H^2(\Gamma)$ even in 3D.

Once $(\delta \tilde{x}_w)$ known, we have to expand these variations overall the mesh. This is done by solving a volumic elasticity system which is also of the form of (2).

As these tools have to be differentiated, it is important to notice that both surfacic and volumic elliptic systems are solved using iterative schemes. This is a real difficulty for AD in reverse mode [8, 7, 4, 29] (see below).

4 Mesh Adaption by Metric Control

We give a brief description of our metric evaluation. The mesh generation technique is not described here. It is based on a Delaunay algorithm [22, 21].

The key idea is to modify the scalar product used in the mesh generator for distance, surface (or volume) evaluations. Hence, the Delaunay approach will generate equilateral elements in a new metric and not in the Euclidean one. This scalar product is based on the evaluation of the Hessian of the variables of the problem. Indeed, for a P^1 Lagrange discretization of a variable u, the interpolation error is bounded by:

$$\mathcal{E} = |u - \Pi_h u|_0 \leq ch^2 |D^2 u|_0, \tag{3}$$

where h is the element size, $\Pi_h u$ the P^1 interpolation of u and $D^2 u$ its Hessian matrix.

This matrix being symmetric, we have:

$$D^2u = \begin{pmatrix} \partial^2u/\partial x^2 & \partial^2u/\partial x\partial y \\ \partial^2u/\partial x\partial y & \partial^2u/\partial y^2 \end{pmatrix}$$

$$= \mathcal{R}\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}\mathcal{R}^{-1},$$

where R is the eigenvectors matrix of D^2u and λ_i its eigenvalues (always real). Using this information, we introduce the following metric tensor $\mathcal{M}$:

$$\mathcal{M} = \mathcal{R}\begin{pmatrix} \tilde{\lambda}_1 & 0 \\ 0 & \tilde{\lambda}_2 \end{pmatrix}\mathcal{R}^{-1}, \tag{4}$$

where

$$\tilde{\lambda}_i = min(max(|\lambda_i|, \frac{1}{h^2_{max}}), \frac{1}{h^2_{min}}),$$

with h_{min} and h_{max} being the minimal and maximal edge lengths allowed in the mesh.

Now, if we generate, by a Delaunay procedure, an equilateral mesh with edges of length of 1 in the metric $\mathcal{M}/(c\mathcal{E})$, the interpolation error $\mathcal{E}$ is equi-distributed over the edges a_i of the mesh. More precisely, we have

$$\frac{1}{c\mathcal{E}}a_i^T M a_i = 1. \tag{5}$$

We notice however that (3) leads to a global error while we would like to have a relative one. We therefore use the following metric definition which takes into account not only the dimension of the variables but also their magnitude:

$$\mathcal{M} = \frac{1}{c\mathcal{E}}\left|\frac{D^2(\Pi_h u)}{\max(|\Pi_h u_h|, \epsilon)}\right|_0, \tag{6}$$

where we have introduced the local value of the variable in the norm. ϵ is a cut-off which defines the admitted difference of the order of magnitude between the larger and smaller scales we are looking for. Bellow ϵ, the phenomena is supposed impossible to be captured by this grid and should therefore be modeled (as in turbulence modeling). This can also be seen as looking for a more precise estimation in regions where the variable is small. Another important consequence of this estimation is that it removes the dimensional problems when intersecting metrics coming from different quantities.

Extension to systems:

For systems of equations, the previous approach leads to a metric for each variable and we take the intersection of all these metrics. Indeed, a metric can be seen through its unit ball which becomes an ellipsoid in case of anisotropy (the unit ball of the Euclidean metric is a sphere). More precisely, for two metrics, we find an approximation of their intersection by the following procedure:

Let λ_i^j and v_i^j, $i,j = 1,2$ the eigen-values and eigen-vectors of $\mathcal{M}_j$, $j = 1,2$. The intersection metric ($\hat{\mathcal{M}}$) is defined by

$$\hat{\mathcal{M}} = \frac{\hat{\mathcal{M}}_1 + \hat{\mathcal{M}}_2}{2}. \tag{7}$$

where $\hat{\mathcal{M}}_1$ (resp. $\hat{\mathcal{M}}_2$) has the same eigen-vectors than $\mathcal{M}_1$, (resp. $\mathcal{M}_2$) but with eigen-values defined by:

$$\tilde{\lambda_i^1} = \max(\lambda_i^1, {v_i^1}^T \mathcal{M}_2 v_i^1), \quad i = 1, 2. \tag{8}$$

The previous algorithm is easy to extend to the case of several variables. As we said, one advantage of (6) is to remove the problem of dimension when intersecting metrics coming from variables with different physical meaning and scale (for instance pressure, density and velocity).

Boundary layers and a priori discretization

The evaluation of the Hessian is done by the Green formula with Neumann boundary condition. This does not lead to a suitable mesh for boundary layers and situations where an a priori minimal discretization along the wall is required. Indeed, the position of the first layer of nodes can be quite irregular (this is important for viscous flows) and the discretization along the wall can become too coarse for a good control space definition.

Another important ingredient therefore, is a mixed Dirichlet-Neumann boundary condition for the metric on wall nodes. More precisely, the eigenvectors for these nodes are the normal and tangent unit vectors and the eigenvalues $(\lambda_\tau, \lambda_n)$ are given by:

$$\lambda_n = max(\frac{1}{h_n^2}, \lambda_n^c), \quad \lambda_\tau = max(\frac{1}{h_\tau^2}, \lambda_\tau^c),$$

where h_n and h_τ are prescribed values to monitor the minimal mesh refinement along the wall in the normal and tangential directions and λ_n^c and λ_τ^c are the computed values (coming from the Hessian). Along the wall the previous metric $\mathcal{M}(x)$ is therefore replaced by a new metric $\hat{\mathcal{M}}(x)$:

$$\hat{\mathcal{M}}(x) = T \Lambda T^{-1},$$

where

$$\Lambda = diag(\lambda_n, \lambda_\tau) \quad \text{and} \quad T = (\vec{n}(x), \vec{\tau}(x)).$$

From a practical point of view, for viscous applications h_n is small and function of the Reynolds number while for inviscid cases h_n is taken large. This metric is slightly propagated through the flow by a smoothing operator.

In the past, we have applied these ingredients to inviscid and viscous laminar and turbulent flows involving shocks and boundary layers. Details on these techniques can be found in [1, 2, 3, 18].

5 Optimization problem

We consider the following minimization problem:

$$min_{x_c} J(x_c, U(x_c)),$$

$$E(x_c, U(x_c)) = 0, \quad g_1(x_c) \leq 0, \quad g_2(U(x_c)) \leq 0,$$

where $x_c \in R^{n_c}$ describe the control parameters, $U \in R^N$ the flow, $E \in R^N$ the state equation and $g_{1,2}$ direct ('geometrical', on x_c) and indirect (on $U(x_c)$, such as a given lift) constraints.

Indirect constraints on U have been taken into account in the cost function by penalty. Geometrical constraints are of two types:

1. shape deformation is allowed between two limiting surfaces (curves in 2D),
2. the volume of the shape can only increase.

The first constraint has been taken into account by projection and the second constraint by penalty in the cost function.

5.1 The Gradient

As we said, in this approach all the nodes on the wall belong to the design space. This makes out of reach an evaluation of the Jacobian by finite difference for instance. Introducing the Lagrangian $L = J + pE$, by the optimality conditions we get:

$$\frac{\partial L}{\partial U} = \frac{\partial J}{\partial U} + p^t \frac{\partial E}{\partial U} = 0, \tag{9}$$

where to get p, we solve:

$$\frac{\partial E^t}{\partial U} p = -\frac{\partial J^t}{\partial U},$$

which has the cost of one flow solution. And after substitution we have:

$$\frac{dJ}{dx_c} = \frac{\partial L}{\partial x_c} = \frac{\partial J}{\partial x_c} + p^t \frac{\partial E}{\partial x_c}.$$

Here the cost is independent of n_c but we still need to compute $\partial E/\partial x_c$. This can be easily done using AD in direct mode for instance [23]. In the present, to avoid developing a particular adjoint solver, we use AD in reverse mode.

5.2 AD in reverse mode

To get the Jacobian of the cost function with respect of control parameters, all the tools described in the previous sections have to be differentiated. But the systems involved are solved using iterative schemes. This is a real problem as the intermediate states have to be stored for the adjoint computation [7, 4].

We use Odyssée developed at INRIA [24, 25, 26]. In [27, 28, 29] the gradients obtained by Odyssée have been compared with those obtained using finite differences in similar contexts. In this approach, the lines of the programs describing the relations between the variation of the design variables and the cost function including the grid and the flow equations are considered as constraints. We associate to each of them a Lagrange multipliers (p) and we construct an augmented Lagrangian (L). The values of the Lagrange multipliers are obtained from the condition that the first variations of L with respect to intermediate variables vanish. The solution can be always obtained simply by back substitution (hence the notion of reverse mode). Once the Lagrange multipliers are evaluated, the gradient of L can be easily calculated. Technical details can be found in [4, 7, 29].

5.2.1 Limitations and keys points

As we said, in reverse mode a dual variable is introduced at each new substitution in the program. This is a problem when the solver contains nested loops (for instance a time integration loop with inside loops over nodes, segments and elements). The limitation of the reverse mode comes therefore from the required memory for Lagrange multipliers and intermediate variables. More details about the limitations we encountered can be found in [4, 7, 8, 29, 28]. For a efficient use of AD in reverse mode in our optimization process, the following key-remarks are essential:

1. For steady flows, the adjoint is independent of the intermediate solutions obtained during time integration. Therefore, we only need to store one state if for a given shape the steady solution is known.

2. Use interprocedural differentiation [26] which consists of replacing what is inside nested loops by routines and differentiating these routines.

3. As we said, to compute the adjoint we always use the steady solution. Therefore, we need to know when the adjoint convergence is sufficient for the optimization to converge. Of course, we can integrate backward for a very long time but this will not be optimal. In fact, it is sufficient for the norm of the gradient of the discrete cost function to be strictly decreasing during optimization [4, 7](n being the optimization iteration):

$$|\nabla J_h^{n+1}| < |\nabla J_h^n|.$$

In other words, when computing the adjoint, we have two different numbers of time step. One for the forward system (taken equal to one) and another for the adjoint system (very large) and we leave the reverse time integration loop once the above criteria is satisfied.

5.3 Adaptive optimization algorithm

The gradient is used in a gradient method to solve the optimization problem. Our minimization tool is quite simple. It is based on a gradient method with a fixed descent step. But more sophisticated methods can be used as far as they do not require the solution of large systems (proportional to n_c). Indeed, the number of nodes on the wall can be quite large, especially in 3D applications (several thousands in [4, 7]). Moreover, as in this approach the number of control variables may vary during optimization due to mesh adaption, the optimization tool should not be sensible to this.

For seek of simplicity, the adaptive optimization algorithm is given with only one optimization after each adaption. But several iterations of optimization can be made on the same mesh. Indeed, we use of the following criteria to detect the time when adaption can be made:

$$J_i^n \leq J_0^n - \epsilon h_{max}^2,$$

where n and i denote respectively the adaption and the associated (on this level) optimization steps and ϵ is free. h_{max} is the maximum segment length allowed in the mesh and we known that all the segments in the mesh have the same length in the local metric M. Moreover, it is interesting to notice that h_{max} is usually chosen once for all. This criteria has to be seen as a sufficient (not necessary) condition for the optimization to converge.

Indeed, we use this criteria together with an a priori maximum number of optimization iterations allowed between two adaptions. A theoretical background for the optimization with a variable discretization and justifications for the previous criteria under more restrictive conditions are available in [30].

At step i of adaption, we denote the mesh, the solution, the metric and the cost $J(x_{ci}, U(x_{ci}))$ by $\mathcal{H}_i$, $\mathcal{S}_i$, $\mathcal{M}_i$ and $J(x_i)$.

Algorithm

$$\mathcal{H}_0, \mathcal{S}_0, \text{ given,}$$

$$\text{do} \quad i = 0, .., iadapt$$

$$\text{define the control space (wall nodes)} : \mathcal{H}_i \rightarrow x_i,$$

$$\text{compute the gradient} : (x_i, \mathcal{H}_i, \mathcal{S}_i) \rightarrow \frac{dJ(x_i)}{dx_i},$$

$$\tilde{x}_i = P_i(x_i - \lambda \frac{dJ(x_i)}{dx_i}),$$

$$\text{deforme the mesh} : (\tilde{x}_i, \mathcal{H}_i) \rightarrow \tilde{\mathcal{H}}_i,$$

$$\text{update the solution over the deformed mesh} : (\tilde{\mathcal{H}}_i, \mathcal{S}_i) \rightarrow \tilde{\mathcal{S}}_i,$$

$$\text{compute the metric} : (\tilde{\mathcal{H}}_i, \tilde{\mathcal{S}}_i) \rightarrow \mathcal{M}_i,$$

$$\text{generate the new mesh using this metric} : (\tilde{\mathcal{H}}_i, \mathcal{M}_i) \rightarrow \mathcal{H}_{i+1},$$

$$\text{interpolate the previous solution over the new mesh} : (\tilde{\mathcal{H}}_i, \tilde{\mathcal{S}}_i, \mathcal{H}_{i+1}) \rightarrow \overline{\mathcal{S}}_{i+1},$$

$$\text{compute the new solution over this mesh} : (\mathcal{H}_{i+1}, \overline{\mathcal{S}}_{i+1}) \rightarrow \mathcal{S}_{i+1},$$

$$\text{end do.}$$

In the previous algorithm, P_i is the projection operator which changes after each adaption as the control space changes and

$$\frac{dJ(x_i)}{dx_i} = \frac{\partial J}{\partial x_i} + (\frac{\partial J}{\partial U})^T (\frac{\partial U}{\partial x_i}).$$

6 Results

In this section we present some results for constrained optimization problems at various Mach number. All these computations have been performed on a workstation making about 10 Megaflop with less than 64 Megabytes of memory in simple precision. These configurations, including mesh adaptions, gradient computations and flow solutions require less than 2 hours. Concerning memory requirements, due to the optimization described in the previous section, the reverse mode requires approximately 10 times more memory than the direct solver. An estimation of the memory required by the direct solver is given by $50 \times N$ words (1 word = 32 or 64 bits depending on architectures), where N is the number of nodes. This is more than what is necessary in a structured solver because that the data structures involved are much more complicated in an unstructured approach.

In the following examples, when global constraints are present, the different penalty coefficients in the cost function are initially chosen for the different quantities involved to have variations of the same order of magnitude. During optimization, they are reduced with the same ratio than the cost function.

The same drag reduction problem with constraints on the lift and the volume of the airfoil has been considered for three different airfoils at different Mach regimes. The cost function for all three cases is given by:

$$J(x) = C_d + \alpha|C_l - C_l^0| + \beta|Vol - Vol_0|,$$

where C_d is the drag coefficient, C_l and C_l^0 are the actual and initial lift coefficients and Vol and Vol_0 the actual and initial airfoil volumes. Our aim is therefore to reduce the drag while the lift and the airfoil volume are kept unchanged. To be sure that the solutions are mesh-independent, a final computation has been done on the final shape until convergence.

The flow solver NSC2KE and the mesh generator BL2D are in free access under `ftp://ftp.inria.fr/INRIA/Projects/Gamma/`.

6.1 Drag reduction for a transonic flow

The initial airfoil is the NACA 0012. The design takes place at Mach number 0.754 and 2 degrees of incidence. The drag coefficient has been reduced by more than 10 percent while the lift and volume slightly increased.

7 Concluding Remarks

A new approach for optimal shape design on variable control spaces and on meshes with variable connectivities has been presented. The reverse mode of automatic differentiation has been used to produce the Jacobian of the discrete operators. In this approach, the mesh is part of the optimization procedure and function of the solution. This is an easy way to avoid mesh dependencies in the optimization as well as in the direct problem.

Acknowledgments

The author would like to thank professor O. Pironneau for his support. The author would like also to thank N. Rostaing-Schmidt and C. Faure of INRIA Sophia Antipolis for their assistance in using Odyssée.

References

[1] H. Borouchaki, M.J. Castro-Diaz, P.L. George, F. Hecht, B. Mohammadi (1996), *Anisotropic adaptive mesh generation in two dimensions for CFD*, 5th Inter. Conf. on Numerical Grid Generation in Computational Field Simulations, Mississipi State Univ.

[2] M. Castro-Diaz, F. Hecht and B. Mohammadi (1995), *"Anisotropic Grid Adaption for Inviscid and Viscous Flows Simulations"*, to appear in JCP.

[3] H. Borouchaki, P.L.George, B. Mohammadi, (1996) *Delaunay Mesh Generation Governed by Metric Specifications. Part II: Applications*, Finite Element in Analysis and Design, special issue on mesh adaption.

[4] B. Mohammadi (1996),*A New Optimal Shape Design Procedure for Inviscid and Viscous Turbulent Flows*, submitted to IJNMF.

[5] O. Pironneau (1984),*Optimal Shape Design for Elliptic Systems*, Springer-Verlag, New York.

[6] A. Jameson (1994),*Optimum Aerodynamic Design via Boundary Control*, AGARD Report 803, Von Karman Institute Courses.

[7] B. Mohammadi (1996),*Optimal Shape Design, Reverse Mode of Automatic Differentiation and Turbulence*, 35 th AIAA conference, Reno.

[8] M. Hafez, B. Mohammadi, O. Pironneau (1996),*Optimum Shape Design using Automatic Differentiation in Reverse Mode* , ICNMFD conference, Monterey.

[9] B.E. Launder and D.B. Spalding (1972),*Mathematical Models of Turbulence*, Academic Press.

[10] B. Mohammadi and O. Pironneau (1994),*Analysis of the K-Epsilon Turbulence Model*, WILEY.

[11] B. Mohammadi, O. Pironneau (1996), *Unsteady Separated Turbulent Flows Computations with Wall-Laws and $k-\varepsilon$ Model*, Submitted to Computer Method in Applied Mechanics and Engineering.

[12] A. Dervieux (1985),*Steady Euler Simulations using Unstructured Meshes*, VKI lecture series, 1884-04.

[13] B. Mohammadi (1994),*CFD with NSC2KE : an User Guide*, Technical report INRIA No.164.

[14] P.L.Roe (1981), *Approximate Riemann Solvers, Parameters Vectors and Difference Schemes*, J.C.P. Vol.43.

[15] G.D.Van Albada, B. Van Leer (1984),*Flux Vector Splitting and Runge-Kutta Methods for the Euler Equations*, ICASE 84-27.

[16] J. Steger, R.F. Warming (1983), *Flux Vector Splitting for the Inviscid gas dynamic with Applications to Finite-Difference Methods*, J. Comp. Phys. 40, pp: 263-293.

[17] S. Osher, S. Chakravarthy (1982), *Upwind Difference Schemes for the Hyperbolic systems of conservation laws*, mathematics of Computation.

[18] F. Hecht, B. Mohammadi (1997),*Mesh Adaption by Metric Control for Multi-scale Phenomena and Turbulence*, submitted to the 35th AIAA, Reno.

[19] D. Vandromme (1983),*Contribution à la modélisation et la prédiction d'écoulements turbulents à masse volumique variable,* Ph.D. thesis, University of Lille.

[20] R. Struijs, H. Deconinck, P. de Palma, P. Roe, G.G.Powel (1991), *Progress on Multidimensional Upwind Euler Solvers for Unstructured Grids,* AIAA paper 91-1550.

[21] P.L. George, F. Hecht, E. Saltel (1990), *Fully automatic mesh generator for 3d domains of any shape,* Impact of Comp. in Sci. and Eng., 2, pp.187-218.

[22] P.L. George (1991), *Automatic mesh generation. Applications to finite element method,* Wiley.

[23] J.M. Malé, B. Mohammadi, N. Rostaing-Schmidt (1996),*Direct and Reverse Modes of Automatic Differentiation of Programs for Inverse Problems: Application to Optimum Shapes Design* Proc. 2nd Int. SIAM Workshop on Computational Differentiation, Santafe.

[24] J.C. Gilbert, G. Le Vey, J. Masse (1991),*La différentiation automatique de fonctions représentées par des programmes,* Rapport de Recherche INRIA 1557.

[25] N. Rostaing-Schmidt (1993), *Différentiation automatique: Application à un problème d'optimisation en météorologie,* Ph.D. Thesis, University of Nice.

[26] C. Faure (1996), *Splitting of Algebraic Expressions for Automatic Differentiation,* In proc. of the Second SIAM Inter. Workshop on Computational Differentiation, Santa Fe.

[27] B. Mohammadi, O. Pironneau (1996),*New Progress in Optimum Shapes Design,* CFD Review 95, Hafez-Oshima Eds, J. Wiley Pub.

[28] B. Mohammadi (1996), *Automatic Differentiation and Nonlinear PDE,* ECCOMAS, Paris.

[29] B. Mohammadi (1996),*Différentiation Automatique par Programme et Optimisation de Formes Aérodynamiques,* MATAPLI 07/96.

[30] E. Polack (1993),*On the Use of Consistent Approximations in the Solutions of Semi-Infinite Optimization and Optimal Control Problems,* Math. Programming, Series B, Vol. 62, No. 2, pp: 385-414.